UNIVERSITY OF ILLINOIS

MAY 19 93

THIS IS TO CERTIFY THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

DERRICK PAUL SCHERTZ

ENTITLED EVALUATING REORDERING SCHEMES FOR FRONTAL METHOD SOLUTIONS

IS APPROVED BY ME AS FULFILLING THIS PART OF THE REQUIREMENTS FOR THE

DEGREE OF BACHELOR OF SCIENCE

Mark S Gaither
Instructor in Charge

APPROVED: R. C. Alkire 1993

HEAD OF DEPARTMENT OF CHEMICAL ENGINEERING

O-1964

ABSTRACT

This thesis describes the algorithm of a FORTRAN code which is used to evaluate reorderings. It computes the number of operations that are required to solve a large, sparse matrix using the Frontal Method. Indirect indications such as solution time or the maximum number of local spikes have been previously used to evaluate reorderings. In contrast, this code provides a direct measure of the performance of a reordering since it counts required operations. Furthermore, the method is quick and thus is easily applied to multiple matrices.

This thesis also compares the performance of different variations of the MNC reversed reordering (Coon 1990) as applied to the frontal method using the program. Min Ratio, cut ratio, three matching heuristics, and three additional switches are explored. The min ratio should be set near 0.3 for regular matrices and the cut ratio does not significantly affect the reordering. Additionally, the reversal of the net definition has a major, yet undefined, affect on the reordering. The reversed P4 reordering was used for comparison and consistently outperformed all versions of MNC.

ACKNOWLEDGEMENTS

I would like to thank Professor Mark A. Stadtherr for the opportunity to do this research and for the guidance and help he has given me. I also thank Kyle V. Camada for his indisposable help in supplying the reordered matrices.
It is through their work and insights that this project developed.

TABLE OF CONTENTS

1. INTRODUCTION

This paper describes the application of FORTRAN code which is used to evaluate the number of operations that will need to be performed during the solution of a matrix using the Frontal Method. Large, sparse matrices arise from process flowsheeting of Chemical Engineering problems. The efficient solution of these matrices is of great interest. One method for improving the solution time involves reordering the rows and columns of the matrix into a form more favorable for the Frontal Method. The code described herein has been used in order to evaluate reorderings in a quick manner, as opposed to actually solving the matrices.

A reordering scheme, which rearranges the rows and columns of a matrix, is typically used to arrange the matrix into a form more favorable to the solver. The Frontal Method (which is a variable bandwidth solver) has been a popular solution method for exploiting the sparse matrix structure while using vector computers. The efficiency of this method depends on the size of the frontal matrix which is generated from the reordered matrix. A good reordering arranges the matrix in a form which leads to a small frontal matrix.

Furthermore, the size of the frontal matrix is directly associated with the maximum number of active local spikes (MLSPK). With the aid of another staistic called row counts, the actual size of the frontal matrix can be traced through a simulated solution. A program which simulates the solution path to find the number of operations is presented and used to judge the relative merit of various reordering schemes, as applied to the frontal method. This program is simple way to evaluate multiple reorderings, as compared with actually solving matrices on a vector computer.

This paper describes the algorithm used in the FORTRAN program and explores its utility. Operation count is compared with the maximum number of local spikes as a measure of the performance of reorderings. The MNC reordering strategy is discussed and variations of the MNC reordering method are explored and compared.

2. A SIMPLIFIED REVIEW OF THE FRONTAL METHOD

This short review of the Frontal Method for equation-based flowsheeting problems is taken largely from Zitney (1993a). It is presented in order to provide a background for the discussion of spikes and row counts. The main algorithm follows:

(1) Assemble a flowsheet equation into the frontal matrix. If all equations have been assembled go to (6).

(2) If there are fully summed variables go to (3), else go to (1).

(3) Choose a pivot element in the fully summed column and normalize the pivot row.

(4) Perform normal Gaussian elimination

(5) Store the pivot row in memory for backsubstitution and remove the pivot row and column from the frontal matrix. Go to (2).

(6) Compute the solution vector by backsubstitution.

The frontal method proceeds by assembling rows into a workspace called the frontal matrix. As a row (equation) is assembled, so are all the variables associated with it. If all the occurences of a given variable have been assembled into the

matrix, the variable is termed fully summed. At this point the variable can be selected as a pivot. Then elimination is used to remove the variable and one equation. Variations on the elimination method exist, however for present purposes we assume normal Gaussian elimination is used. In all methods the frontal matrix is regarded as dense; that is, all elements are assumed to be nonzero. The coefficients of the eliminated equation are saved for later backsubstitution. After each new row is assembled eliminations are performed repeatedly until no more eligible pivots remain and then another row is assembled. In this manner the entire matrix is assembled and eliminated, leaving a series of simple backsubstitutions to be completed. An detailed example of the frontal matrix is found in Appendix A.

3. SPIKES AND ROW COUNTS

Two statistics which are based solely on the nonzero structure of the matrix are of interest, spikes and row counts. Neither are new concepts. Spikes have been used to evaluate the merit of reorderings. Row count is a value that is used in some soltuion programs. Both statistics are formed as arrays, the indices being the rows of the matrix.

Spikes refer to variables that have nonzero elements above the diagonal. We say that a column is spiked for a given row if 1) an element of the column occurs above the diagonal, and 2) an element of the column occurs at or above the given row. These are the elements which create the need for pivoting. A matrix without spikes (a lower triangular matrix) could be solved by a series of simple substitutions. Counting spikes provides useful information about the number of rows that need to be assembled before eliminations can occur, and how many

eliminations can occur at each step. The maximum number of local spikes (MLSPK) has been used judge the merit of reordering schemes.

Row counts describe how the width of the frontal matrix changes as rows are added. The row count for a given row is defined as the number of variables which have their last occurrence (in a column) in the given row. In other words, it is the number of fully summed variables which will exist in the frontal matrix after a given row is assembled. The sum of row counts equals the number of rows.

Figure 1 helps to clarify these characteristics. The spike elements are denoted by 'S' and '.', and the significant row count elements are underlined.

Table 1. Example of Spikes and Row Counts

	1	2	3	4	5	6	7	8	9	0	Spikes	Row Count
1	X										1	0
2		X									1	0
3			X		S						2	0
4			X		X						1	1
5					X	X	.		S		3	0
6	X					X	X		.		2	0
7	X						X	X	.		2	0
8	X				X				X	S	2	2
9					X	X	X		X	X	1	4
10						X				X	0	3.

4. PROGRAM ALGORITHM

An example from Zitney (1993a) is followed through in detail. Special attention is paid to the spikes and row counts; in particular why and how they affect the size of the frontal matrix. The original unreordered matrix is:

	1	2	3	4	5	6
1	X	X				X
2		X	X	X		
3			X	X		
4				X	X	X
5	X			X		X
6		X	X	X	X	X

And the reordered matrix:

	3	2	5	6	1	4
3	X					X
2	X	X				X
4			X	X		X
6	X	X	X	X		X
1		X		X	X	
5				X	X	X

At this point it is informative to inspect the matrix for spikes and row counts. The elements which create spikes are denoted by 'S' while the last element in each column (for row counts) is underlined. The number of underlined elements for a given row is defined as its row count. The rows and columns have been renumbered:

	1	2	3	4	5	6	Spikes	Row Count
1	X					S	1	0
2	X	X				S	1	0
3			X	S		S	2	0
4	X	X	X	X		S	1	2
5		X		X	X	.	1	1
6				X	X	X	0	3

Note that the column 4 spike which is present in row 3 has been absorbed in row 4, and that row 5 is considered spiked in column 6. The spike and row count vectors comprise all of the information needed to estimate the operation count.

The operation count is formulated as follows:

1. Each time an equation is assembled, add one to the number of rows, and add the row count (associated with the incoming row) to the number of columns.

2. Each time an elimination is performed, subtract one from both the number of rows and number of columns.

3. The number of eliminations which can be performed (the number of fully summed variables) after assembling a row is equal to (dSpike + 1), where dSpike is the decrease in spikes between the row previously assembled and the row most recently assembled. The spike count for the zeroth row is taken to be zero, as the matrix starts with no spikes. For example, we will see that two eliminations can be performed after row 4 is assembled, since

Spike(3) - Spike(4) + 1 = 2 - 1 + 1 = 2.

Before solving, the matrix is flipped corner for corner in order to approximate an upper triangular matrix. The spike and row count vectors likewise are flipped. This in effect reverses the reordering. We will now apply the operation count rules. Renumbering the rows and columns again for convenience, we have:

	1	2	3	4	5	6	Spikes	Row Count
1	X	X	X				1	3
2		X	X		X		1	1
3	S		X	X	X	X	2	2
4	S		S	X			1	0
5	S				X	X	1	0
6	S					X	0	0

Assembly of the first equation leads to:

	1	2	3
1	X	X	X

The matrix grows to 1 x 3 (0 + 1, 0 + 3) and the number of eliminations expected is zero (0 - 1 + 1). Now equation 2 is assembled, bringing along variable 5:

	1	2	3	5
1	X	X	X	
2		X	X	X

The size grows to 2 x 4 (1 + 1, 3 + 1) and one elimination (1 - 1 + 1) is possible, leaving:

```
    1   3   5
1   X   X   F
```

where the 'F' stands for fill, a nonzero created during an elimination. The frontal size has been reduced to 1 x 3 (2 - 1, 4 - 1). Assembling equation 3 leads to:

```
    1   3   5   4   6
1   X   X   F
3   X   X   X   X   X
```

The dimensions reach 2 x 5 (1 + 1, 3 + 2) while no eligible pivots exist (1 - 2 + 1 = 0) and thus the next equation is assembled (3 x 5):

```
    1   3   5   4   6
1   X   X   F
3   X   X   X   X   X
4   X   X       X
```

Variables 3 and 4 are now both fully summed and eligible as pivots. This is reflected as in rule 3 above (2 - 1 + 1 = 2), as two eliminations are possible. Equation 1 can be used to eliminate variable 3, and the dimensions become 2 x 4:

```
    1   5   4   6
3   X   X   X   X
4   X   F   X
```

Then equation 3 can be used to eliminate variable 4, with the dimensions shrinking to 1 x 3:

```
    1   5   6
4   X   F   F
```

The asssembly of equation 5 makes variable 5 fully summed (1 - 1 + 1 = 1, 2 x 3):

```
     1   5   6
4    X   F   F
5    X   X   X
```

And its elimination leads to (1 x 2):

```
     1   6
4    X   F
```

The assembly of the last equation yields (1 - 0 + 1 = 2, 2 x 2):

```
     1   6
4    X   F
6    X   X
```

And pivoting on element (1,1) gives (1 x 1):

```
     6
6    X
```

The final pivot is element (6,6) and the problem is completed using backsubstitution with the stored (eliminated) rows. Thus the combination of spike and row count vectors provides enough information to predict the history of the frontal matrix dimensions during a solution. Tables 1 and 2 summarize the process in a more concise form:

Table 2. Spike and Row Count arrays

N	Spikes	Row Count
1	1	3
2	1	1
3	2	2
4	1	0
5	1	0
6	0	0

Table 3. History of the Frontal Matrix

Act	New Dim	dRow	dCol	Eliminations
Assemble 1	1 x 3	+ 1	+ 3	0 - 1 + 1 = 0
Assemble 2	2 x 4	+ 1	+ 1	1 - 1 + 1 = 1
Eliminate	1 x 3	- 1	- 1	
Assemble 3	2 x 5	+ 1	+ 2	1 - 2 + 1 = 0
Assemble 4	3 x 5	+ 1	+ 0	2 - 1 + 1 = 2
Eliminate	2 x 4	- 1	- 1	
Eliminate	1 x 3	- 1	- 1	
Assemble 5	2 x 3	+ 1	+ 0	1 - 1 + 1 = 1
Eliminate	1 x 2	- 1	- 1	
Assemble 6	2 x 2	+ 1	+ 0	1 - 0 + 1 = 2
Eliminate	1 x 1	- 1	- 1	
Eliminate	0 x 0	- 1	- 1	

In order to estimate the operation count, we must examine the frontal matrix dimensions at the times eliminations take place . Since the chart above lists the new dimensions after eliminations, we look instead at the entries immediately preceeding eliminations. It is these dimension upon which the eliminations were performed.

Given the dimensions of a frontal matrix, we claim that the number of operations needed to eliminate a row and column is related to the area of the matrix, that is, number of rows times number of columns. This is the number of multiplications which are required by Gaussian elimination.

Two important statistics can be considered. First, the maximum operation count describes the largest single frontal matrix area which was encountered. This is similar to the maximum number of local spikes since it characterizes only the largest frontal matrix. The maximum frontal area also describes the amount of space needed to store the frontal matrix. The other, more accurate criterion is the

sum of the individual frontal areas. This is the total number of operations which need to be performed in order to solve the whole matrix.

5. NOTES ON THE PROGRAM

The matrix (reordered or unreordered) is read in using the Integer Column Index (ICI) and Integer Row Pointer (IRP) format (Appendix B). This format uses two vectors to specify the locations of nonzero elements. Thc first array, ICI(1..NZ), lists the column indexes of the nonzero elements, reading from left to right starting at the first row of the matrix. There are NZ entries, where NZ is the total number of nonzero elements in the matrix. The other array, IRP(1..N+1), contains row pointers which indicate the ICI index of the element which begins each row. IRP(N+1) is assigned equal to (NZ+1) in order to aid referencing purposes.

All program operations are based on this format. Because the ICI array is not automatically ordered within a row for some reordering schemes, a sorting routine is immediately performed to insure that they are in ascending order. It should also be noted that the indexing of elements in the code is less than straightforward. This is due to the indirect ICI and IRP format.

The spike array elements are first initialized to zero, and then the first major loop is executed to generate the spike array. The reordered matrix is then reversed corner for corner in preparation for the solution simulation. Since the array is being flipped, the spike array must also be reversed so that the spike counts correspond to the same rows. A second extended loop tallies the row counts on this reversed array. (The choice to evaluate spikes before reversal and row counts after reversal is arbitrary.) After the spike and row count arrays have been completed, the final

extended loop tracks the frontal matrix size as the equations are assembled and eliminated. This counting is done exactly as described in the previous section.

Since the matrix is flipped before the simulation is performed, it essentially transforms the original reordering into a reversed reordering. In other words, if a P4 reordered matrix is input, the analyses applies to the reversed P4 reordering. All of the reorderings considered in this paper have been reversed.

6. REORDERING METHOD DESCRIPTIONS

MNC Strategy

The MNC reordering method (Coon) is designed to break the matrix into independent partitions which can then be solved efficiently on a parallel computer using various solution methods. The reorderings are based on a bipartite graph representation of the matrix, where the vertices correspond to the rows and columns and edges correspond to the locations of non-zero off-diagonal elements.

To break up the matrix into independent partitions, the MNC reorderings attempt to find a complete matching of the row and column vertices of a subset of the matrix, while examining the number of nets cut by the partition. A net is defined as a column vertex together with the row vertices with which it shares an edge in the bipartite graph.

MNC Options

Two basic parameters are input into the reorderings; min ratio is a tolerance of variation in the relative partition sizes, and cut ratio is a tolerance for the number of rows cut relative to the size of the set being partitioned.

Three heuristics are used to determine which pair of vertices to move across a partition boundary to minimize the number of nets cut by the partition;

1. Free-match criterion. Pair the vertex at the top of the list with its match in the current matching, and determine if it should be moved.

2. Free-swap criterion. Pair the vertex at the top of the list with a free vertex in a cycle of arbitrary length four in the other set.

3. Maximum gain exchange criterion. In this case both lists are searched until a cycle is found containing two vertices such that no edges are cut. This involves much more searching than either heuristics 1. or 2.

Three additional switches must be set. Sometimes none of the criteria are satisfied for a vertex at the top of the list. So first, the algorithm must specify whether vertices can reenter the list later. Secondly, the main algorithm is designed to produce equal size blocks for maximum parallelism, so if one partition cannot be further divided, no other partition is allowed to be divided smaller than the irreducible one. This balance rule can be maintained or relaxed. And finally, some variations use a reversed net definition in an attempt to adapt MNC to variable bandwidth solver such as the frontal method. The reversed net is defined as a row vertex together with the column vertices with which it shares an edge.

P4 Reordering

P4 (Hellerman 1972) is a classic reordering scheme which examines the equations and variables locally to reorder them. It makes no effort to form equation blocks or to apply theory (such as MNC's graph theory) to determine the

reordering. Instead it seeks to form a lower triangular matrix with the minimum number of spikes. It has proven effected, but has also been surpassed in efficiency. It is included for comparison purposes.

7. RESULTS AND DISCUSSION

The matrices and reorderings were evaluated using the operation count (sum of the frontal matrix areas) and the maximum frontal area statistics. The frontal matrix size history for several reorderings were also graphed. In general, good reorderings lead to low operation counts and small frontal matrix maximums.

A total of five matrices were examined. The first three were generated by the Harwell-Boeing method and are of relatively small size. The other two are from linked distillation flowsheets and are somewhat larger. In addition to having natural structure, they have been previously reordered by HGCS. Nine reordering schemes were tested. Seven are based on the MNC reordering method (Coon 1990). Unreordered matrices and matrices reordered by P4 were also tested.

The versions of MNC tried were of course a subset of all the possible combinations of heuristics and definitions in the previous section. The seven reorderings tested were chosen based on experiments and recommendations from Coon (1990) and are listed in Table 3. The min ratio was varied at 0.1, 0.3, and 0.5, while the cut ratio was varied at 0.25, 0.35, and 0.45. These nine combinations were tried for each of the seven MNC variations.

Table 4. MNC Variations Tested

	Heuristics	Reentry	Balanced	Net Defn
1	Free-match, then Free-swap	No	Yes	Normal
2	Free-match, then Free-swap	Yes	Yes	Normal
3	Free-match, then Free-swap	No	No	Normal
4	Free-match, then Free-swap	Yes	No	Normal
5	Free-match, then Max gain exch	Yes	Yes	Normal
6	Free-match, then Free-swap	Yes	Yes	Reversed
7	Free-match, then Free-swap	Yes	No	Reversed

The MNC performance was compared to the unreordered matrix for each of the variations. The cut ratio did not seem to effect the reordering performances and is set at 0.25 for the following examples. However, Figures 1 and 2 illustrate the importance of the min ratio. The x-axis indicates the elimination step number. The y-axis indicates the frontal matrix area during a given elimination. The sum of these areas is the total operation count and is listed. The min ratio of 0.3 clearly provides the best reordering. This min ratio value led to equivalent or better reorderings than the other min ratios tested in almost every case.

Figures 1 and 2 also give some insight on the utility of the operation count. Other indicators consider only the maximum frontal area, or in the case MLSPK, the maximum frontal height. While these both give some clue to the overall operation count, the shape of the curves vary and prevent maximum statistics from being consistantly accurate indicators. If the shape of the curves shown in Figures 1 and 2 could be accurately predicted, the maximum frontal area would become an accurate indicator. Here it is not.

Some interesting behavior is shown in Figure 3. This plot compares different min ratios for Problem 4 using the MNC3 reordering version. The structure

inherent to this test problem can been seen in the regular, sawtooth curve. This is caused by the periodicy encountered in the equations describing the different trays of the linked distillation from which the problem arises. The 'unreordered' frontal matrix history shows that the combination of inherent structure and the HGCS reordering has been extremely successful at maintaining a small frontal matrix size. Applying the MNC3 reordering merely breaks up the structure and increases the operation count. The P4 reordering improves only slightly on the original matrix. In this case, a min ratio of 0.1 actually outperforms the min ratio of 0.3, but since this problem is atypical (and with the aid of other data) we maintain that 0.3 is a better choice overall for general problems.

All the tested variations of MNC for problem three are plotted in Figure 4. A min ratio of 0.3 has been assumed. Also plotted for comparison are the unreordered matrix and the P4 matrix. MNC improves the operation counts greatly, but not by as much as P4. The MNC variations are divided into two distinct groups. The is because the first five variations use a normal net definition while MNC 6 and 7 use a reversed net definition.

Figure 5 again illustrates the interesting behavior of matrix problem 4. When MNC reorderings are applied, this inherent structure is broken up and the reordering actually makes the problem more difficult to solve. Again, the first five MNC reorderings with the normal net definition differ significantly from MNC 6 and 7 with the reversed net definition. The P4 reordering improved this matrix only slightly.

Expected Solution Times

The differences encountered in the operation counts in these experiments are

large but not critical. Since all the matrices were relatively small (17,82,132,154,2310) the overhead time needed to set up the elimination steps is much larger than the amount of time spent in the actual eliminations. This overhead is proportional to N, the order of the matrix, and we can write the time needed to solve the matrix as 'aN + Operation Count'. Since the complexity of a matrix grows much faster than N, the operation count will become a critical contribution to the solution time as the matrices become truly large. It is for these truly large matrices that the reordering programs are critical.

8. CONCLUSIONS

The FORTRAN program presented makes a quick and accurate evaluation of the merit of a reordering. It does so by inspecting spikes and row counts and then simulating the assembly of the frontal matrix. A graph of the frontal area may be generated for more detailed information.

Amongst the 7 MNC variations analyzed there was only a slight difference; the biggest difference was caused by the reversal of the net definition.
The most important MNC parameter is the min ratio, and indications show that 0.3 may be a reasonable value. The P4 reordering outperformed all MNC variations. Problems 4 and 5 of the test set showed incredible structure.

It should be reemphasized that since so few matrices were explored, this research serves 1) rather to set up the program and analysis method, and 2) to explore some of the MNC parameters, rather than to provide any conclusive evidence concerning the performance of MNC with the frontal method of solution.

However, this program does provide quick and accurate insight on the merit of reordering algorithms, and can even provide a method of visualizing the

development of the frontal matrix. It is expected to be highly useful in the task of adapting MNC to work for frontal method solutions.

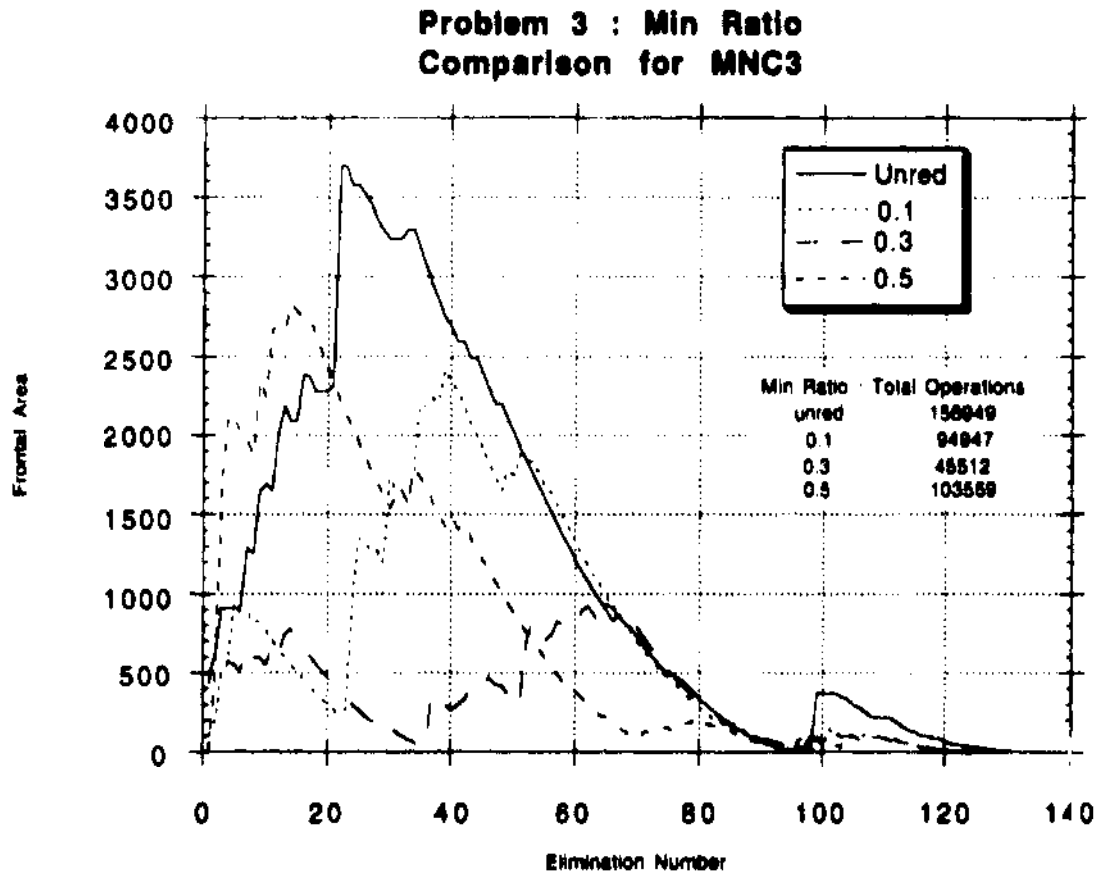

Figure 1. Min Ratio Comparison for MNC3, Problem 3

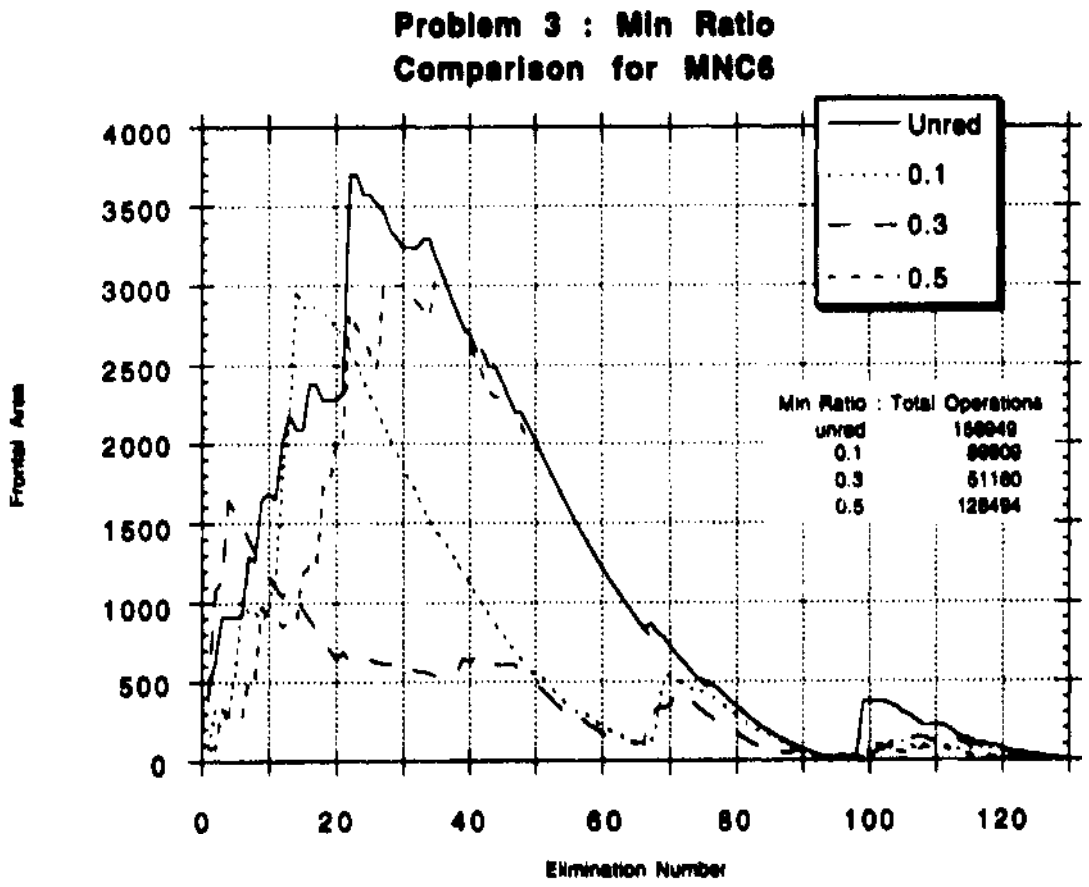

Figure 2. Min Ratio Comparison for MNC6, Problem 3

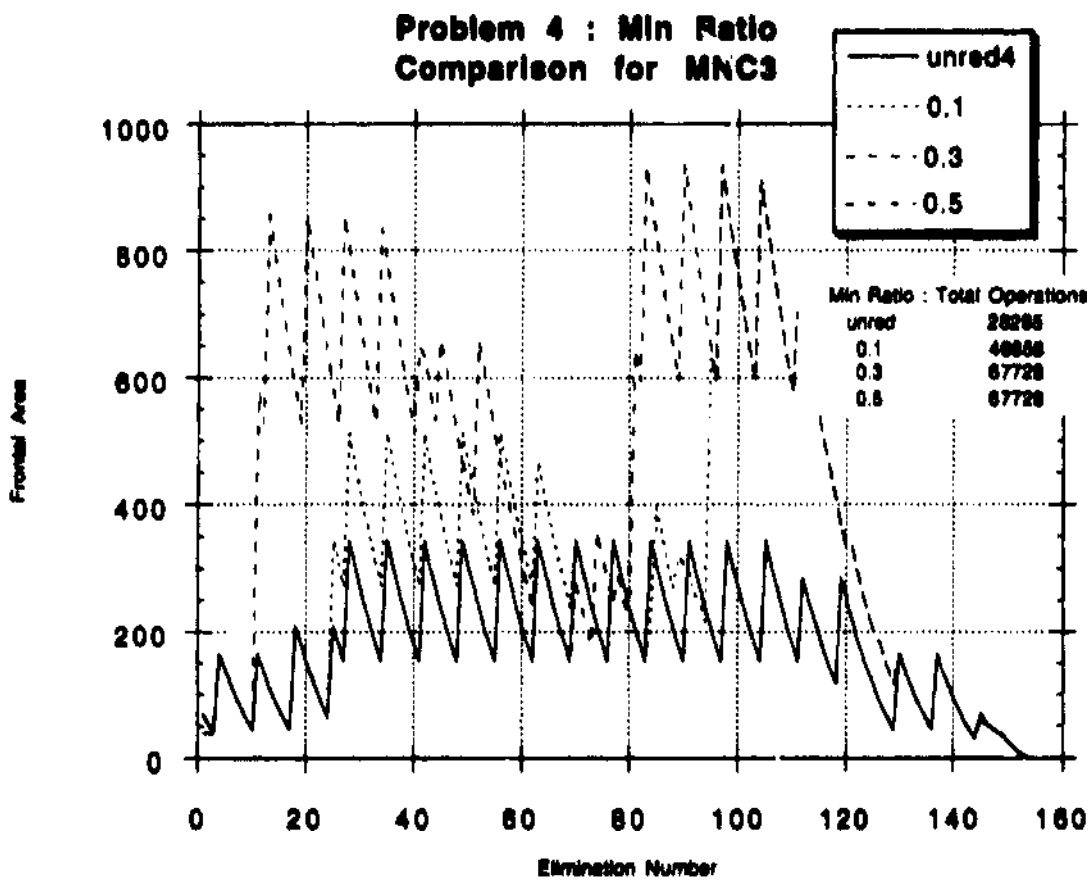

Figure 3. Min Ratio Comparison for MNC3, Problem 4

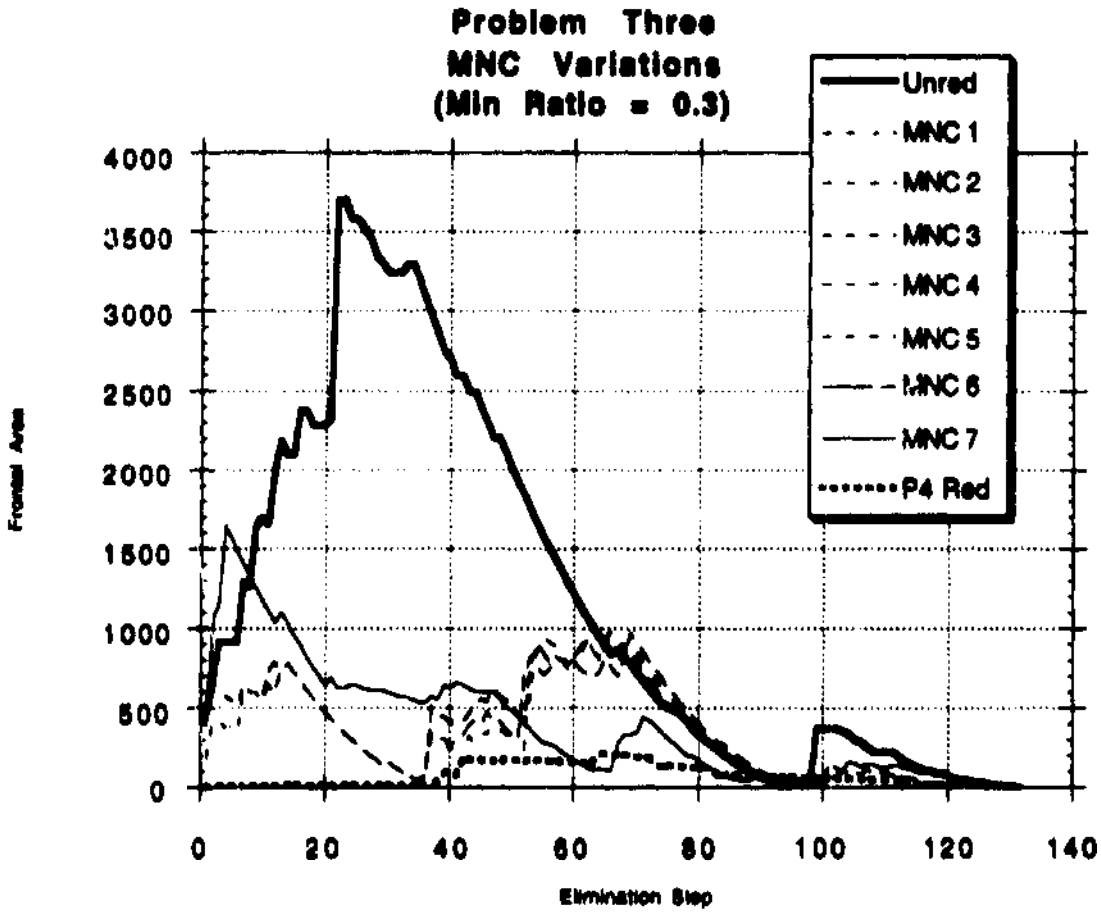

Figure 4. MNC Variations on Problem Three

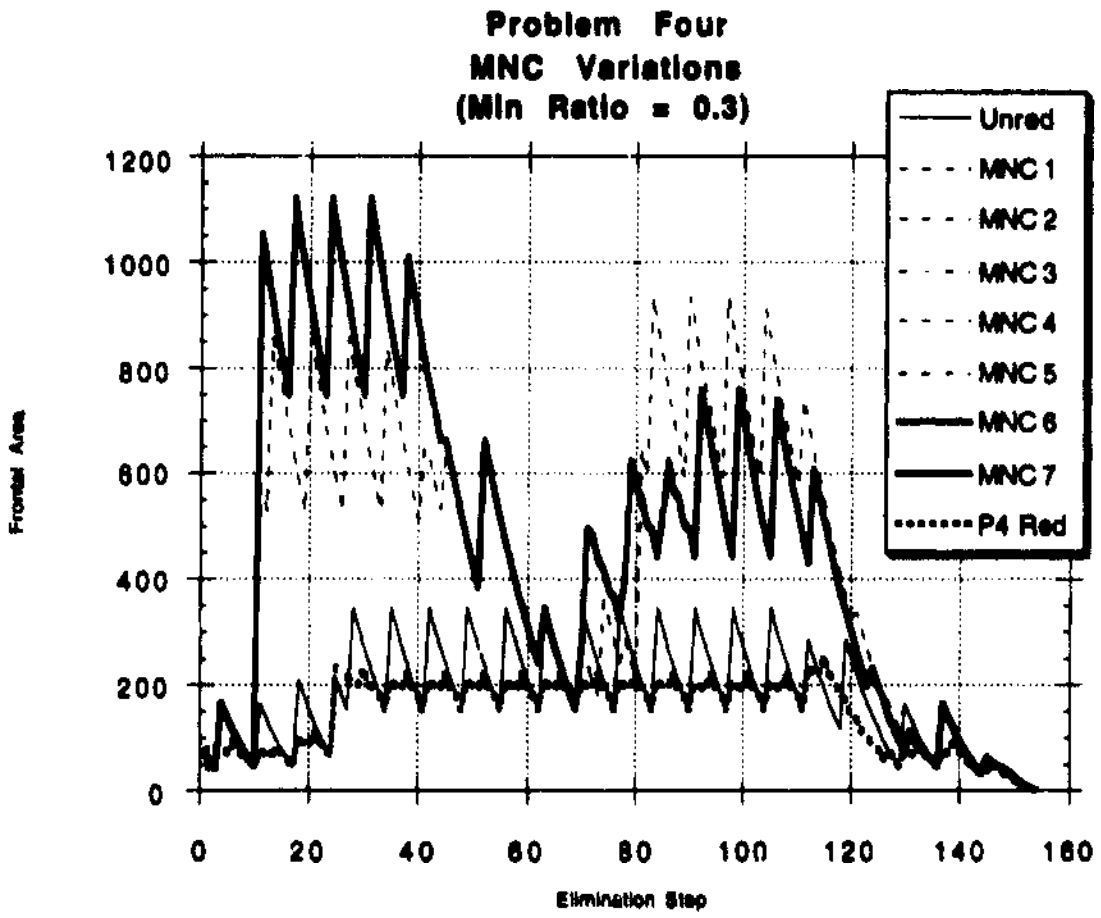

Figure 5. MNC Variations on Problem Four

REFERENCES

Coon, Alan B. and Mark A. Stadtherr. Generalized Block-Tridiagonal Matrix Orderings for Parallel Computation in Process Flowsheeting. Submitted for Publication. Department of Chemical Engineering, University of Illinois at Urbana-Champaign.

Coon, Alan B. PhD Thesis. Ordering and Dirrect Methods for Coarse Granular Parallel Solutions in Equation-Based Flowsheeting. Department of Chemical Engineering, University of Illinois at Urbana-Champaign, 1990.

Hellerman, E. and D. Rarick. The Partitioned Preassigned Pivot Procedure (P4). In "Sparse Matrices and Their Applications". Eds. D. J. Ross and R. A. Willoughby. Plenum Press: New York, 1972. p. 76.

Zitney, Stephen E. and Mark A. Stadtherr. Frontal Algorithms for Equation-Based Chemical Process Flowsheeting on Vector and Parallel Computers. Computers chem. Engng. Vol. 17, No. 4, 319-338 (1993a).

Zitney, Stephen E. and Mark A. Stadtherr. Supercomputing Strategies for the Design and Analysis of Complex Separation Systems. Ind. Eng. Chem. Res. 32, 604-612 (1993b).

APPENDICES

APPENDIX A

THE BASIC FRONTAL METHOD

Zitney's (1993a) basic example is followed through. The implications of spikes and row counts are not presented.

The original matrix given below has not been reordered:

	1	2	3	4	5	6
1	X	X				X
2		X	X	X		
3			X	X		
4				X	X	X
5	X			X		X
6		X	X	X	X	X

Once a reordering has been applied, the matrix might look like the one below. The row and column numbers have been maintained to illustrate the movement:

	3	2	5	6	1	4
3	X					X
2	X	X				X
4			X	X		X
6	X	X	X	X		X
1		X		X	X	
5				X	X	X

In order to solve the matrix, it is flipped corner for corner. Renumbering the rows and columns again for convenience, we then have:

	1	2	3	4	5	6
1	X	X	X			
2		X	X		X	
3	X		X	X	X	X
4	X		X	X		
5	X				X	X
6	X					X

As prescribed by the frontal method algorithm, the first equation and its variables are assembled into the frontal matrix. Only variables 1, 2, and 3 are active:

	1	2	3
1	X	X	X

Since none of the variables are fully summed, no eliminations can occur and equation 2 is assembled, bringing along variable 5:

	1	2	3	5
1	X	X	X	
2		X	X	X

At this point variable 2 is fully summed and a pivot can be chosen. Either of the two elements could be chosen; we will use (2,2). Row two is normalized and stored in memory, then row 2 and column 2 are eliminated leaving:

	1	3	5
1	X	X	F

During the elimination phase, element (1,5) will take on a nonzero value. This is referred to as a fill-in, and is denoted by F. At this point no variables are fully summed, so equation 3 is assembled:

	1	3	5	4	6
1	X	X	F		
3	X	X	X	X	X

Again no eligible pivots exist and thus the next equation is assembled:

	1	3	5	4	6
1	X	X	F		
3	X	X	X	X	X
4	X	X		X	

Variables 3 and 4 are now both fully summed and eligible as pivots. Equation 1 can be used to eliminate variable 3:

	1	5	4	6
3	X	X	X	X
4	X	F	X	

Then equation 3 can be used to eliminate variable 4:

	1	5	6
4	X	F	F

The asssembly of equation 5 makes variable 5 fully summed:

	1	5	6
4	X	F	F
5	X	X	X

And its elimination leads to:

	1	6
4	X	F

The assembly of the last equation yields:

	1	6
4	X	F
6	X	X

And pivoting on element (1,1) gives:

	6
6	X

The final pivot is element (6,6) and the problem is completed using backsubstitution.

APPENDIX B

THE FORTRAN PROGRAM

```
C         1         2         3         4         5         6         7
C23456*890123456789012345678901234567890123456789012345678901234567890123456789012
```

```
C  Derrick P. Schertz
C  University of Illinois at Urbana-Champaign
C  Chemical Engineering 292
C  Undergraduate Senior Research

C  In Cooperation with...
C  Professor Mark A. Stadtherr and Kyle V. Camarda
C  Department of Chemical Engineering

C  May 1993

C=========================
C  Program to determine the operation count for a large, sparse matrix
C  Data format is currently set to the style generated by MNC reorderings

C  simscan3.f : Data Files must be in this format:
C    line of junk
C    line of junk
C    j j n nz j    <j=JUNK, ignore this stuff>
C    IRP(1..N+1)
C    ICI(1..NZ)

C  Final Printout :   MAXFROW
C                     MAXFCOL
C                     MAXOPER
C                     MOR x MOC
C                     OPER
C                     MLSPK
C                     N
C                     ROWSPK(I) using an Implicit DO

C  Program reads arrays but does not change them in any way.
C    This is done in anticipation of incorporating this code into a program
C    which would already have the ICI(LICI) and IRP(N) matrices.
```

```
C The file to be read in should contain the nonzero elements of the matrix in two C
C       arrays:
C ICI(I) is an integer array that contains the column indices of
C  the nonzero elements of the coefficient matrix, stored row by row.

C IRP(I) is an integer array that points to the start of each row in
C  the ICI(I) array.  IRP(I) is the position in the ICI(I) array
C  where data for row I begins.

C Input file is 'mat1' and output file is 'width.out'

C NLSPK current number of active local spikes
C MLSPK maximum number of concurrent spikes that has occurred
C SPIKE(I) indicates whether a spike is active in column I
C ROWSPK(I) number of active local spikes at row I
C MAXOPER area of the frontal matrix at its largest dimension
C OPER sum of MAXOPER over the entire elimination procedure
C MOR, MOC dimensions of frontal matrix at largest area (MAXOPER)
C MAXFCOL maximum number of columns in the frontal matrix
C MAXFROW maximum number of row in the frontal matrix (=MLSPK + 1)

C Note that in its current form, this code accomodates a 5000 square
C   matrix with up to 20000 elements.  This should be changed as necessary.

C=========================

    PROGRAM SPK

    INTEGER N,LICI
    INTEGER SPIKE(5000),ROWSPK(5001),NLSPK,MLSPK,I,J,K
    INTEGER IRP(5001),IRP1,IRP2,ICI(20000),ICI1
    INTEGER KIRP(5001),KICI(20000),IRPX
    INTEGER ICITEMP,MIN
    INTEGER RC(5001),POKE(5001)
    INTEGER FCOL,FROW,TEMP,OPER
    INTEGER MAXOPER,MAXFCOL,MAXFROW,MOC,MOR

    CHARACTER*7 JA
C               For handling the junk strings in the input file

    OPEN(4,FILE='mat1')
```

```
      OPEN(5,FILE='width.out')

      NLSPK = 0
      MLSPK = 0

C  Input N, LICI, arrays ICI(LICI) and IRP(N) using Implicit DOs
         READ(4,*) JA
         READ(4,*) J
         READ(4,*) JA,J,N,LICI,J
         READ(4,*) JA,JA
         READ(4,*) (IRP(I),I=1,N+1)
         READ(4,*) (ICI(I),I=1,LICI)

C  Sorting Routine ... to ensure ICI are ordered within a row
      DO 51 I=1,N
      DO 57 J=IRP(I),IRP(I+1)-1
       MIN=ICI(J)
       DO 58 K=J+1,IRP(I+1)-1
           IF(ICI(K).LT.MIN) THEN
            ICITEMP=ICI(K)
          ICI(K)=MIN
             MIN=ICITEMP
         ENDIF
58     CONTINUE
       ICI(J)=MIN
57     CONTINUE
51     CONTINUE

C  Initialize SPIKE(I) to zero: start with no spikes
      DO 10 I=1,N
       SPIKE(I) = 0
10     CONTINUE

      IRP2=IRP(1)

C=========================
      DO 20 I=1,N

C  Undo spike as pointer moves down a row and spike is absorbed
       IF (SPIKE(I).EQ.1) THEN
        SPIKE(I) = 0
        NLSPK = NLSPK - 1
```

```
      ENDIF

       IRP1=IRP2
       IRP2=IRP(I+1)

          DO 30 J=IRP1,IRP2-1

          ICI1=ICI(J)

C  Mark a column as spiked if 1) nonzero element 2) not already spiked
           IF ((ICI1.GT.I).AND.(SPIKE(ICI1).EQ.0)) THEN
             SPIKE(ICI1) = 1
             NLSPK = NLSPK + 1

C  Update maximum spike counter MLSPK, ROWSPK
             IF (NLSPK.GT.MLSPK) THEN
               MLSPK = NLSPK
             ENDIF
           ENDIF
30      CONTINUE

        ROWSPK(I)=NLSPK
20    CONTINUE
C=========================
        ROWSPK(N+1)=0


C    **** REVERSAL SECTION ****

C  SAVE TO A SECOND SET OF ARRAYS
      DO 61 I=1,LICI
          KICI(I)=ICI(I)
61    CONTINUE
      DO 62 I=1,N+1
          KIRP(I)=IRP(I)
62    CONTINUE

C  DO THE ACTUAL REVERSAL
      DO 64 I=1,LICI
          ICI(I)=N+1-KICI(LICI+1-I)
64    CONTINUE
```

```
      IRP(1)=1
      IRPX=0
      DO 65 I=2,N+1
         IRP(I)=IRP(I-1)+KIRP(N+3-I)-KIRP(N+2-I)
65    CONTINUE
      IRP(N+1)=LICI+1

C  REVERSE THE ROWSPK ARRAY ALSO
      DO 70 I=1,N/2
         TEMP=ROWSPK(I)
         ROWSPK(I)=ROWSPK(N+1-I)
         ROWSPK(N+1-I)=TEMP
70    CONTINUE

C     **** RC COUNTING SECTION ****

      DO 72 I=1,N
         RC(I)=0
72    CONTINUE
      K=0
      POKES=0

      DO 74 J=1,LICI                    'for each ICI value;
         IF (POKE(ICI(J)).EQ.0) THEN    'if this column has not been assigned
          POKE(ICI(J))=1                'it will now be assigned
76       IF (IRP(K).LE.J) THEN         'increment K to find corresponding row
           K=K+1
           GOTO 76
          ENDIF
          RC(K-1)=RC(K-1)+1             'credit this variable to row K-1
       ENDIF
74    CONTINUE

C  Frontal Matrix Size Counting Routine ...
C     Major arguments are RC(I) and ROWSPK(I)

      FCOL=0
      FROW=0
      OPER=0
      MAXOPER=0
      MAXFCOL=0
      MAXFROW=0
```

```
      MOC=0
      MOR=0
      DO 84 I=1,N
          FCOL=FCOL+RC(I)
          FROW=FROW+1
          IF ((ROWSPK(I)+1).GT.ROWSPK(I+1)) THEN
           DO 86 J=1,ROWSPK(I)+1-ROWSPK(I+1)
            OPER=OPER+FCOL*FROW
             IF (FCOL*FROW.GT.MAXOPER) THEN
                 MAXOPER=FCOL*FROW
                 MOC=FCOL
                 MOR=FROW
             ENDIF
             IF (FCOL.GT.MAXFCOL) THEN
              MAXFCOL=FCOL
             ENDIF
             IF (FROW.GT.MAXFROW) THEN
              MAXFROW=FROW
             ENDIF

            FCOL=FCOL-1
              FROW=FROW-1
      86    CONTINUE
             ENDIF
      84   CONTINUE

      C  REVERSE THE ROWSPK ARRAY FOR PRINTING PURPOSES
          DO 92 I=1,N/2
             TEMP=ROWSPK(I)
             ROWSPK(I)=ROWSPK(N+1-I)
             ROWSPK(N+1-I)=TEMP
      92   CONTINUE

      C    PRINTOUT TO FILE 5
             WRITE(5,*) 'Maximum Frontal Column Height : ',MAXFROW
             WRITE(5,*) 'Maximum Frontal Column Width : ',MAXFCOL
             WRITE(5,*) 'Maximum Frontal Area(Operations) : ',MAXOPER
             WRITE(5,*)
             WRITE(5,*) 'Max Dimension Frontal (RxC) : ',MOR,'x',MOC
             WRITE(5,*) 'Number of Operations Required : ',OPER
             WRITE(5,*) 'Max Number of Local Spikes : ',MLSPK
             WRITE(5,*) 'Order of Array : ',N
```

```
      WRITE(5,*)
      WRITE(5,*) (ROWSPK(I),I=1,N)

C   PRINTOUT TO SCREEN
      WRITE(*,*) 'Maximum Area : ',MOR,' x ',MOC,' = ',MAXOPER
      WRITE(*,*) 'Max Sizes : ',MAXFROW, MAXFCOL, OPER
      WRITE(*,*) 'Operation Count : ',OPER

    END
```

APPENDIX C

FURTHER INFORMATION ON THE PROGRAM

1. The input format may be easily adapted to input permutation vectors, such as those produced by P4 reorderings. The following lines of code should be used:

```
      INTEGER ORN(5000),OCN(5001),NCN(5001),PMICI(20000),PMIRP(5001)
      INTEGER ROW,KRP,KCI
...
C  Input with permutation vectors and transformation
      READ(4,*) N,LICI
      READ(4,*) (PMICI(I),I=1,LICI)
      READ(4,*) (PMIRP(I),I=1,N)
      PMIRP(N+1)=LICI+1

C  Read ORN and OCN arrays
      READ(4,*) (ORN(I),I=1,N)
      READ(4,*) (OCN(I),I=1,N)

      DO 32 I=1,N
        ORN(I)=IABS(ORN(I))
        OCN(I)=IABS(OCN(I))
32    CONTINUE

C  Create NCN array
      DO 39 I=1,N
       NCN(OCN(I))=I
39    CONTINUE

C  Convert (permutation vectors, PMICI, PMIRP) to (ICI, IRP)
      KRP=0
      KCI=0
      DO 41 ROW=1,N
C          ** Place next row into first available position
        KRP=KRP+1
        IRP(KRP)=KCI+1
C          ** Last filled position(PMICI) + 1 = 1st position in new row
        DO 42 J=PMIRP(ORN(ROW)),PMIRP(ORN(ROW)+1)-1
```

```
            KCI=KCI+1
            ICI(KCI)=NCN(PMICI(J))
42        CONTINUE
41      CONTINUE
        KRP=KRP+1
        IRP(KRP)=KCI+1
C                ** Record final IRP value : IRP(N+1)
```

2. To perform an operation count on a non-reversed reordering, all reversal sections should be removed and the row count should be redefined to reflect the number of variables which have their first occurence (in a column) in the given row.

3. For graphing purposes it is useful to print the frontal matrix areas to the output file in a column, removing the text and other values. This facilitates moving the numbers to a graphing spreadsheet, such as Kaleidagraph. In this case the output to the file 5 should be replaced with:

```
        DO 94 I=1,N
        WRITE(5,*) ROWSPK(I)
94      CONTINUE .
```